A High Level Cryptographic API for PKI-Enabled Applications

Shu-jen Chang
NIST
shu-jen.chang@nist.gov

September 13, 2001

Overview

- Needs for a High Level API
- NIST CAPI
- Implementation Status
- Agency/Industry Adoption
- Future Work

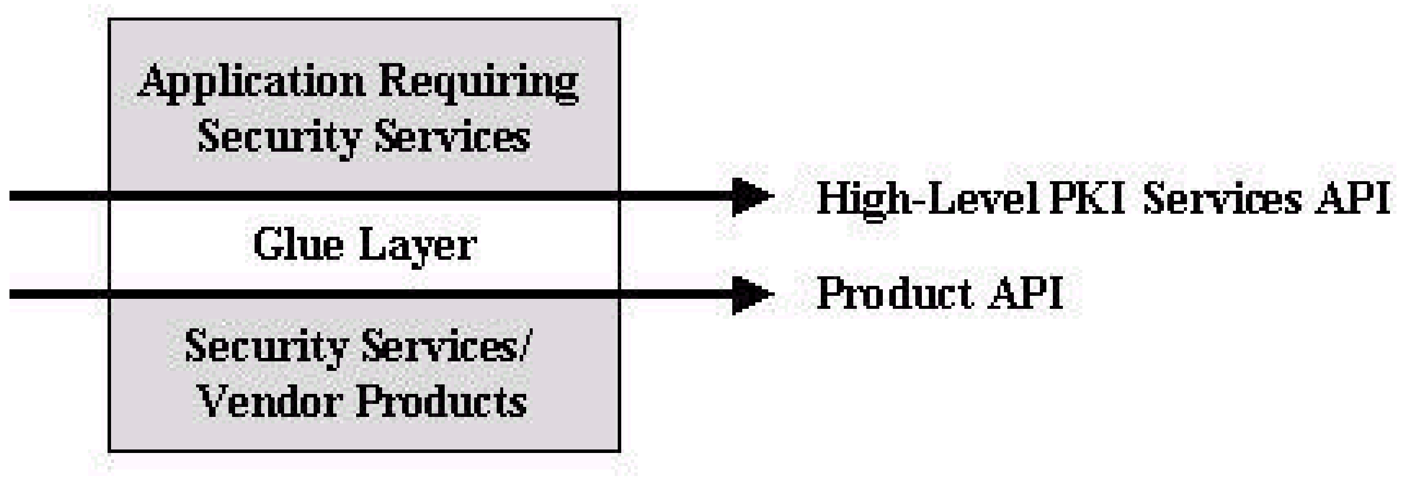
The Challenge

- Multiplicity of Cryptographic APIs
- High Programmer Turnover Rate
- Vendor Support
- Complexity of Code
  - Few understand the technology
  - Too easy to make mistakes
  - Complexity impedes development and analysis of PKI applications

Benefits

- Easier to code, analyze, audit, maintain
- Provides better control
- Vendor neutral
- Easier to port applications since underneath services have been checked
- Facilitates PKI application development
- Forces the PKI to do the heavy lifting

PKI Services API


Application Requiring Security Services
Glue Layer
Security Services/ Vendor Products
High-Level PKI Services API
Product API

NIST High Level CAPI

- SignBuffer
- VerifyBuffer
- EncryptBuffer
- DecryptBuffer
- CMSBufferParser

- SignFile
- VerifyFile
- EncryptFile
- DecryptFile
- CMSFileParser

PKI Support

- User login/logout
- Repository access, certificate/CRL retrieval
- Maintains user states, configuration tables/files
- PKI Does the heavy lifting

API Assumptions

- All calls made on behalf of one user identity (one key) at a time
- Application allocates/deallocates memory for I/O parameters
- Each function returns a return code and additional error message
- Use data type SignedData and EnvelopedData defined in CMS (RFC2630)

Signature Generation - SignBuffer

```
int signBuffer(
        IN          uint32         data_length,
        IN          char*          data_to_sign,
        IN          Boolean        authent_required,
        IN          Boolean        encap_data_flag,
        IN/OUT      uint32*        signed_data_length,
        IN/OUT      SignedData*    signed_data,
        OUT         char*          error_data
);
```

Signing Operation

- Reauthenticate the user if requested
- Locate the signer's key - prompt user to login if not already logged in
- Generate digital signature over data provided
- Package signature and other information in CMS SignedData format
- Return success or error code to application

Signing Operation

- One signature for each invocation, repeat calls for multiple signatures
- Receiving party usually receives all the information needed for signature verification, though signed content can be omitted from resulting signature structure (SignedData). In this case, application should maintain association between content signed and the output SignedData.

SignedData

- **CMS version**
- **Digest algorithm**
- **Encapsulated content**
- **Certificates (Optional)**
- **CRLs (Optional)**
- **Signer Information**

Signature Generation - SignFile

```
int signFile (
        IN          char*          infile,
        IN          Boolean        authent_required,
        IN          Boolean        encap_data_flag,
        IN          Boolean        output_to_file,
        IN          char*          outfile,
        IN/OUT      uint32*        signed_data_length,
        IN/OUT      SignedData*    signed_data,
        OUT         char*          error_data
);
```

Signature Verification

```
int verifyBuffer(
    IN        uint32              signed_data_length,
    IN        SignedData*         signed_data,
    IN        ushort              policy,
    IN        uint32              data_length,
    IN        char*               data_to_verify,
    OUT       char*               signer,
    OUT       GeneralizedTime     time_data_signed,
    IN/OUT    uint32*             output_data_length,
    IN/OUT    char*               output_data,
    OUT       char*               error_data
    );
```

Signature Verification

- Parse DER-encoded signature data, check required policy if any, verify signature
- Return the following if signed_data is parseable or signature is verifed:
  - signer informtion
  - time data was signed
  - data that was signed
  - error message (if verification failed)

VerifyFile

```
int verifyFile (
        IN      char*               file_signed,
        IN      char*               signature_file,
        IN      ushort              policy,
        OUT     char*               signer,
        OUT     GeneralizedTime     time_data_signed,
        OUT     char*               error_data
);
```

Buffer Encryption

```
int encryptBuffer (
    IN          char**            recipientlist,
    IN          uint32            data_length,
    IN          char*             data_to_encrypt,
    IN          ushort            encryption_algorithm,
    IN          Boolean           authent_required,
    IN/OUT      uint32*           enveloped_data_length,
    IN/OUT      EnvelopedData*    enveloped_data,
    OUT         char*             error_data
);
```

Encryption

- A message may be encrypted for multiple recipients with one call using the same symmetric encryption algorithm
- Operation is more complicated than signing due to key management choices
  - key agreement
  - key transport
  - key encrypting key

Encrypt Operation

- Locate & validate recipient's encryption/key agreement certificate
- Generate session (symmetric) key, protect it using selected key management mechanism
- Encrypt data buffer under the session key using specified encryption algorithm
- Encode ciphertext and needed information for recipients to decrypt in EnvelopedData

EncryptFile

```
int encryptFile (
    IN        char**           recipientlist,
    IN        char*            file_to_encrypt,
    IN        ushort           encryption_algorithm,
    IN        Boolean          authent_required,
    IN        Boolean          output_to_file,
    IN        char*            encrypted_file,
    IN/OUT    uint32*          enveloped_data_length,
    IN/OUT    EnvelopedData*   enveloped_data,
    OUT       char*            error_data
);
```

EnvelopedData

- **CMS version**
- **Originator info OPTIONAL**
- **Recipient infos**
- **Encrypted content info**
  - **Content type**
  - **Content encryption algorithm identifier**
  - **Encrypted content OPTIONAL**
- **Unprotected attributes OPTIONAL**

Buffer Decryption

```
int decryptBuffer (
    IN          uint32                  enveloped_data_length,
    IN          EnvelopedData*          enveloped_data,
    IN          Boolean                 authent_required,
    IN/OUT      uint32*                 plain_text_length,
    IN/OUT      char*                   plain_text,
    OUT         char*                   sender,
    OUT         ushort*                 encryption_algorithm,
    OUT         char*                   error_data
);
```

Decrypt Operation

- Enveloped_data contains everything the recipients will need for decryption
- Authent_required flag
  - Used only if key agreement/transport is involved
  - Indicates whether to reauthenticate before a user can use his private key to decrypt protected key
- Returns plain text, sender information, encryption algorithm, or error message

File Decryption

```
int decryptFile (
IN      char*           encrypted_file,
IN      Boolean         authent_required,
IN      char*           plain_text_file,
OUT     char*           sender,
OUT     ushort*         encryption_algorithm,
OUT     char*           error_data
);
```

CMS Parser

- Not a crypto. function, but useful to have
- Similar to the S/MIME feature
- Allows application to obtain signer information and signed content without signature verification
- Returns signer information, time data was signed, signed content, or error message

CMSBufferParser

```
int CMSBufferParser (
    IN          uint32              signed_data_length,
    IN          SignedData*         signed_data,
    OUT         char*               signer,
    OUT         GeneralizedTime     time_data_signed,
    IN/OUT      uint32*             content_length,
    IN/OUT      char*               content_signed,
    OUT         char*               error_data
    );
```

CMSFileParser

```
int NIST_CMSFileParser (
    IN      char *              signature_file,
    OUT     char*               signer,
    OUT     GeneralizedTime     time_data_signed,
    OUT     char*               error_data
);
```

API Status

- API specification available at: http://csrc.nist.gov/pki/pkiapi/welcome.htm
- Currently under review
- Implementation started at NIST and FDIC

FDIC Implementation

- Ongoing implementation built on top of Entrust Toolkit
- Possible second implementation on top of MS CAPI
- A high assurance financial application will be developed to use NIST CAPI for GAO sanctioning

NIST Implementation

- Built on top of S/MIME Freeware Library (SFL) and CML
  - http://www.getronicsgov.com/hot/sfl_home.htm
  - http://www.getronicsgov.com/hot/cml_home.htm
- Started in summer 2001
- Currently finishing code for signing and verification

Agency/Industry Adoption

- Several agencies are interested in NIST CAPI for their PKI applications
- Many have legacy applications to transistion to use new technologies
- Assistance is needed for such transitions
- Agencies are participating/modeling after the FDIC effort
- Industry buy-in more difficult but desirable

Future Work

- Functions to support CA/RA operations
  - PKI Specifications to Support the DOE Travel Manager Program, August 1996
- Mechanism to support web-based applications

More Information

- NIST PKI API page: http://csrc.nist.gov/pki/pkiapi/welcome.htm

- Send comments to:
  - **Shu-jen Chang (shu-jen.chang@nist.gov)**